第3章　递　　推

递推算法(recurrence algorithm)是一种应用非常广泛的常用算法,与第4章的递归有着非常密切的联系。

本章探讨递推在求解数式、数列、数阵以及计数等诸多案例方面的应用。

3.1　递推概述

在纷繁变幻的世界,所有事物都随时间的流逝发生着微妙的变化。许多现象的变化是有规律可循的,这种规律往往呈现出前因后果的关系。某种现象的变化结果与紧靠它前面变化的一个或一些结果紧密关联,递推的思想正体现了这一变化规律。

3.1.1　递推算法

所谓递推,是在命题归纳时,可以由 $n-k$,$n-k+1$,…,$n-1$ 的情形推得 n 的情形。一个线性递推可以形式地写成

$$a_n=c_1a_{n-1}+c_2a_{n-2}+\cdots+c_ka_{n-k}+f(n)$$

其中,$f(n)=0$ 时递推是齐次的,否则是非齐次的。递推的一般解法要用到 n 次方程的求根。

递推算法是利用问题本身所具有的递推关系求解问题的一种方法。设要求问题规模为 n 的解,当 $n=1$ 时,解或为已知,或能非常方便地得到解。能采用递推法构造算法的递推性质,能从已求得的规模为 1,2,…,$i-1$ 的一系列解,构造出问题规模为 i 的解。这样,程序可从 $i=0$ 或 $i=1$ 出发,重复地由已知解至 $i-1$ 规模的解,通过递推,获得规模为 i 的解,直至得到规模为 n 的解。

递推算法的基本思想是把一个复杂、庞大的计算过程转换为简单过程的多次重复,该算法充分利用了计算机的运算速度快和不知疲倦的特点,从头开始一步步地推出问题最终的结果。使用递推算法编程,既可使程序简练,又可节省计算时间。

对于一个序列来说,如果已知它的通项公式,那么要求出数列第 n 项或求数列的前 n 项之和是简单的。但是,在许多情况下,要得到数列的通项公式是困难的,有时甚至无法得到。然而,一个有规律的数列,其相邻位置的数据项之间通常存在一定的关系,可以借助已知的项,利用特定的关系逐项推算出它的后继项的值,直到找到所需的那一项为止。递推算法避开了求通项公式的麻烦,把一个复杂问题的求解分解成若干步简单运算。

递推算法的首要问题是得到相邻的数据项之间的关系,即递推关系。它针对这样一类问题:问题的解决可以分为若干步骤,每个步骤都产生一个子解(部分结果),每个子解都是由前面若干子解生成的。我们把这种由前面的子解得出后面的子解的规则称为递推关系。

递推关系是一种高效的数学模型,是组合数学中的一个重要解题方法,在组合计数中有

着广泛的应用。在概率方面利用递推可以解决一类基本事件个数较大的概率问题。在对多项式的求解过程中，很多情况可以使用递推算法来实现。在行列式方面，某些 n 阶行列式只用初等变换难以解决，但如果采用递推求解则显得较为容易。

递推关系不仅在各数学分支中发挥着重要的作用，由它所体现出来的递推思想在各学科领域中更是显示出其独特的魅力。

3.1.2　递推实施步骤与描述

在设计求解问题前，要通过细心地观察，丰富地联想，不断尝试推理，尽可能归纳总结其内在规律，然后再把这种规律性的东西抽象成递推数学模型。

利用递推求解问题，需要掌握递推关系的具体描述及其实施步骤。

1. 实施递推的步骤

1）确定递推变量

应用递推算法解决问题，要根据问题的具体情况设置递推变量。递推变量可以是简单变量，也可以是一维或多维数组。

2）建立递推关系

递推关系是指如何从变量的前一些值推出其下一个值或从变量的后一些值推出其上一个值的公式（或关系）。

递推关系是递推的依据，是解决递推问题的关键。

有些问题，其递推关系是明确的，大多数实际问题并没有现成的明确的递推关系，需根据问题的具体情况，不断尝试推理，才能确定问题的递推关系。

3）确定初始（边界）条件

对所确定的递推变量，要根据问题最简单情形的数据确定递推变量的初始（边界）值，这是递推的基础。

4）对递推过程进行控制

递推过程不能无休止地执行下去。递推过程在什么时候结束，满足什么条件结束，这是递推算法必须考虑的递推过程控制问题。

递推过程的控制通常可分为两种情形：一种是所需的递推次数是确定的值，可以计算出来；另一种是所需的递推次数无法确定。对于前一种情况，可以构建一个固定次数的循环来实现对递推过程的控制；对于后一种情况，需要进一步根据问题的具体情况归纳出用来结束递推过程的条件。

2. 递推算法框架描述

递推通常由循环来实现，一般在循环外确定初始（边界）条件，在设置的循环中实施递推。

下面归纳常用的递推模式并做简要的框架描述。

递推按流向可分为顺推与逆推。

1）简单顺推算法

顺推即从前往后推，从已求得的规模为 $1,2,\cdots,i-1$ 的一系列解，推出问题规模为 i 的解，直至得到规模为 n 的解。

简单顺推算法框架描述：

```
f(1～i-1)=<初始值>;                    //确定初始值
for(k=i;k<=n;k++);
   f(k)=<递推关系式>;                   //根据递推关系实施顺推
print(f(n));                           //输出 n 规模的解 f(n)
```

2) 简单逆推算法

逆推即从后往前推,从已求得的规模为 n,$n-1$,…,$i+1$ 的一系列解,推出问题规模为 i 的解,直至得到规模为 1 的解。

简单逆推算法框架描述:

```
f(n～i+1)=<初始值>;                    //确定初始值
for(k=i;k>=1;k--)
   f(k)=<递推关系式>;                   //根据递推关系实施逆推
print(f(1));                           //输出解 f(1)
```

3) 二维数组顺推算法

简单递推问题设置一维数组实现,较复杂的递推问题需设置二维或二维以上数组。

设递推的二维数组为 $f(k,j)$,$1\leqslant k\leqslant n$,$1\leqslant j\leqslant m$,由初始条件分别求得 $f(1,1)$,$f(1,2)$,…,$f(1,m)$,需求 $f(n,m)$,则据给定的递推关系由初始条件依次顺推得 $f(2,1)$,$f(2,2)$,…,$f(2,m)$;$f(3,1)$,$f(3,2)$,…,$f(3,m)$……直至得到所要求的解 $f(n,m)$。

二维数组顺推算法框架描述:

```
f(1,1 : m)=<初始值>;                   //赋初始值
for(k=2;k<=n;k++)
   for(j=1;j<=m;j++)
     f(k,j)=<递推关系式>;               //根据递推关系实施递推
print(f(n,m));                         //输出 n 规模的解 f(n,m)
```

4) 多关系分级递推算法

当递推关系包含两个或两个以上关系式时,通常应用多关系分级递推算法求解。

多关系分级递推算法框架描述:

```
f(1: i-1)=<初始值>;                    //赋初始值
for(k=i;k<=n;k++)
{  if(<条件 1>)
     f(k)=<递推关系式 1>;              //根据递推关系 1 实施递推
   ⋮
   if(<条件 m>)
     f(k)=<递推关系式 m>;              //根据递推关系 m 实施递推
}
print(f(n));                           //输出 n 规模的解 f(n)
```

关于递推的时间复杂度,如果在一重循环中可完成递推,通常其相应的时间复杂度为 $O(n)$。在实际应用中,由于递推关系的不同,往往需要二重或更复杂的循环结构才能完成递推,其相应的时间复杂度为 $O(n^2)$或更高。

3.2 超级素数搜索

第 2 章的全素组与素数幻方都是有关素数的案例,本节进一步探索素数中一个新的子集——超级素数。

1. 案例提出

定义 $m(m>1)$位超级素数：

(1) m 位超级素数本身为素数。

(2) 从高位开始，去掉 1 位后为 $m-1$ 位素数；去掉 2 位后为 $m-2$ 位素数……去掉 $m-1$ 位后为 1 位素数。

例如，137 是一个 3 位超级素数，因 137 是一个 3 位素数，去高 1 位得 37，是一个 2 位素数；去高 2 位得 7，是一个 1 位素数。而素数 107 不是超级素数，因去高 1 位得 7 不是一个 2 位素数。

输入整数 $m(1<m\leqslant 10)$，统计 m 位超级素数的个数，并输出其中最大的 m 位超级素数。

下面试应用枚举与递推两种算法设计求解。

2. 枚举设计

1) 设计要点

(1) 为了方便判别素数，应用试商法设计素数判别函数 $p(k)$：若 k 为素数，$p(k)$返回 1；否则，$p(k)$返回 0。

(2) 为枚举 m 位数需要，通过自乘 10(即 $c=c*10$;)，计算 m 位数的起始数 c。

(3) 设置枚举 m 位奇数的 f 循环。

① 若 f 不是素数，或 f 的个位数字不是 3 或 7(超级素数的个位数字必然是 3 或 7)，则返回。

② 若 f 的其他各位数字出现 0，显然应予排除。

③ 除 m 位数 f 本身及其个位数已检验外，从高位开始去掉 1 位，2 位，…，$m-2$ 位可得 $m-2$ 个数($f\%k$，$k=100,1000,\cdots,10^{m-1}$)，这 $m-2$ 个数的 p 函数值相乘为 t：

若 $t=0$，说明 $m-2$ 个数中至少有一个非素数，则返回。

若 $t=1$，说明 $m-2$ 个数全为素数，应用变量作统计个数。

④ 为输出最大的 m 位超级素数，在统计的同时，作赋值：“e=f;”，最后输出的 e 则为最大的 m 位超级素数。

2) 程序实现

```
//枚举求指定 m 位超级素数,c321
#include<stdio.h>
#include<math.h>
void main()
{  int i,m;
   long c,d,e,f,k,s,t;
   int p(long f);
   printf("  请确定 m(m>1): "); scanf("%d",&m);
   for(c=1,i=1;i<=m-1;i++)
     c=c*10;                          //确定最小的 m 位数 c
   s=0;
   for(f=c+1;f<=10*c-1;f=f+2)         //设置枚举循环,f 为 m 位奇数
   {  if(p(f)==0 || !(f%10==3 || f%10==7)) continue;
      for(t=1,d=f/10,i=1;i<=m-2;i++)
```

```
    {  if(d%10==0) t=0;                                    //枚举中间 m-2 个数字
       d=d/10;
    }
    if(t==0) continue;
    for(t=1,k=10,i=1;i<=m-2;i++)
      { k=k*10;t=t*p(f%k);}                                //枚举 m-2 次去位操作
    if(t==0) continue;
    s++;e=f;                                               //统计并赋值
  }
  printf("  共%ld个%d位超级素数。\n",s,m);
  printf("  其中最大数为%ld。\n",e);
  }
  #include<math.h>
  int p(long k)                                            //设计素数检测函数
  {  int j,h,z;
     z=0;
     if(k==2) z=1;
     if(k>=3 && k%2==1)
     {  for(h=0,j=3;j<=sqrt(k);j+=2)
          if(k%j==0) {h=1;break;}
        if(h==0) z=1;                                      //k 为素数返回 1,否则返回 0
     }
     return z;
  }
```

3) 程序运行示例

```
请确定 m(m>1): 5
共 192 个 5 位超级素数。
其中最大数为 99643。
```

3. 递推设计

1) 设计要点

根据超级素数的定义,m 位超级素数去掉高位数字后是 $m-1$ 位超级素数。一般地,$k(k=2,3,\cdots,m)$位超级素数去掉高位数字后是 $k-1$ 位超级素数。

那么,在已求得 g 个 $k-1$ 位超级素数 $a[i](i=1,2,\cdots,g)$时,在 $a[i]$的高位加上一个数字 $j(j=1,2,\cdots,9)$,得到 $9g$ 个 k 位候选数 $f=j\times e[k]+a[i](e[k]=10^{k-1})$,只要对这 $9g$ 个 k 位候选数检测即可。这就是从 $k-1$ 递推到 k 的递推关系。

注意到超级 $m(m>1)$位素数的个位数字必然是 3 或 7,则得递推的初始(边界)条件:

```
a[1]=3,a[2]=7,g=2;
```

2) 程序实现

```
//递推求指定 m 位超级素数,c322
#include<stdio.h>
#include<math.h>
void main()
{  int g,i,j,k,m,t,s;
   double d,f,a[20000],b[20000],e[20];
   int p(double f);
   printf("  请确定 m(m>1): "); scanf("%d",&m);
```

```
    g=2;s=0;
    a[1]=3;a[2]=7;e[1]=1;                    //递推的初始条件
    for(k=2;k<=m;k++)
    {  e[k]=e[k-1]*10;t=0;
       for(j=1;j<=9;j++)
          for(i=1;i<=g;i++)
          {  f=j*e[k]+a[i];                  //产生 9g 个候选数 f
             if(p(f)==1)
             { t++;b[t]=f;
               if(k==m) {s++;d=f;}           //统计并记录最大超级素数
             }
          }
       g=t;
       for(i=1;i<=g;i++) a[i]=b[i];          //g 个 k 位 b[i]赋值给 a[i]
    }
    printf("  共%d个%d位超级素数。\n",s,m);
    printf("  其中最大数为%.0f。\n",d);
}
int p(double k)
{  int h,z;double j;long t;
   z=0;
   t=(int)pow(k,0.5);
   for(h=0,j=3;j<=t;j+=2)
       if(fmod(k,j)==0) {h=1;break;}
   if(h==0) z=1;                             //如果 k 为素数返回 1,否则返回 0
   return z;
}
```

3）程序运行示例

```
请确定 m(m>1): 9
共 545 个 9 位超级素数。
其中最大数为 999962683。
```

4. 两个算法的时间复杂度比较

应用枚举设计，需对 m 位奇数进行检测，枚举数量级为 10^m。

应用递推设计，只需检测 $9g(k-1)$次（$g(k-1)$为 $k-1$ 位超级素数的个数），$k=2,3,\cdots,m$，因而求 m 位超级素数共检测的次数为

$$s(m)=9\sum_{k=1}^{m-1}g(k)$$

其中，$s(m)$是对递推设计的时间复杂度的定量估算，其数量级远低于 10^m，即递推设计的时间复杂度要低于枚举设计。

例如，当 $m=5$ 时，应用枚举设计，调用检测函数 $p(k)$的次数为 $45\,000\times4=180\,000$；而应用递推设计，调用 $p(k)$函数的次数仅为 $9\times(2+11+39+99)=1359$。

3.3　递 推 数 列

本节探讨两个典型的递推数列案例的求解，一个是新颖的摆动数列，另一个是涉及分子、分母分别递推的分数数列。

3.3.1 摆动数列

1. 案例提出

已知递推数列:

$$a(1)=1,a(2i)=a(i)+1,a(2i+1)=a(i)+a(i+1) \quad (i \text{ 为正整数})$$

试求该数列的第 n 项与前 n 项中哪些项最大,最大值为多少。

2. 递推设计

该数列根据项序号为奇或偶两种情况做不同递推,所得数列各项呈大小有规律的摆动。

设置 a 数组,赋初值 $a[1]=1$。根据递推式,在 i 循环中根据项序号 $i(2\sim n)$ 为奇或偶做不同递推:

(1) $\mathrm{mod}(i,2)=0$(即 i 为偶数)时,$a[i]=a[i/2]+1$。

(2) $\mathrm{mod}(i,2)=1$(即 i 为奇数)时,$a[i]=a[(i+1)/2]+a[(i-1)/2]$。

每得一项 $a[i]$,与最大值 max 做比较,如果 $a[i]>$max,则 max$=a[i]$。

在得到最大值 max 后,在所有 n 项中搜索哪些项为最大项(因最大项可能多于一项),并输出最大值 max 及所有搜索得到的最大项。

3. 程序设计

```
//摆动数列,c331
#include<stdio.h>
void main()
{  int i,n,max,a[10000];
   printf("  请确定项数 n: ");
   scanf("%d",&n);
   a[1]=1;max=0;
   for(i=2;i<=n;i++)
   {  if(i%2==0)                    //分情况实施递推
        a[i]=a[i/2]+1;
      else
        a[i]=a[(i-1)/2]+a[(i+1)/2];
      if(a[i]>max) max=a[i];        //比较得最大值
   }
   printf("  a(%d)=%d \n",n,a[n]);
   printf("  摆动数列前%d项中最大项有: ",n);
   for(i=2;i<=n;i++)                //探索所有的最大项
     if(a[i]==max) printf("a(%d)=",i);
      printf("%d \n",max);
}
```

4. 程序运行示例与分析

```
请确定项数 n: 2015
a(2015)=131
摆动数列前 2015 项中最大项有: a(1707)=a(1877)=321
```

上述递推在一重循环中完成,时间复杂度为 $O(n)$。

本案例求最大项的处理是新颖的,先求出最大值,然后逐项比较求出各个最大项。这一处理方式对要求求出所有最大(或最小)项时具有示范意义。

3.3.2　分数数列

1. 案例提出

一个递推分数数列的前 6 项如下：

$$1/2,3/5,4/7,6/10,8/13,9/15,\cdots$$

该数列的构成规律是第 i 项的分母 d 与分子 c 存在关系 $d=c+i$，而分子 c 为与前 $i-1$ 项中的所有分子、分母均不相同的最小正整数。

对于指定的正整数 n，试求出该数列的第 n 项，并求出前 n 项中的最大项。

2. 设计要点

注意到递推需用到前面的所有项，设置数组 $c[i]$表示第 i 项的分子，$d[i]$表示第 i 项的分母(均为整数)。显然，初始值为：$c[1]=1$，$d[1]=2$；$c[2]=3$，$d[2]=5$。

已知前 $i-1$ 项，如何确定 $c[i]$呢？

显然 $c[i]>c[i-1]$，同时可证，当 $i>2$ 时，第 i 个分数的分子 $c[i]$总小于第 $i-1$ 个分数的分母 $d[i-1]$。

于是，置 k 在区间$(c[i-1],d[i-1])$取值，k 分别与 $d[1],d[2],\cdots,d[i-1]$比较，若存在相同的数，则 k 增 1 后再比较；若没有相同的数，则产生第 i 项，进行赋值：$c[i]=k$，$d[i]=k+i$。

为了准确求出数列前 n 项中的最大项，设最大项为第 k_{max}项(k_{max}赋初值 1)，每产生第 i 项，如果有

$$c[i]/d[i]>c[k_{max}]/d[k_{max}]$$

则有

$$c[i]\times d[k_{max}]>c[k_{max}]\times d[i]$$

即第 i 项要比原最大的第 k_{max}项大，则赋值 $k_{max}=i$，把产生的第 i 项确定为最大项。产生第 n 项后，前 n 项中的最大项也比较出来了。

在程序设计中比较最大项，应用整数不等式是适宜的。

3. 程序设计

```
//分数递推数列,c332
#include<stdio.h>
void main()
{  int n,i,k,t,j,kmax;
   long c[3001],d[3001];
   printf("请输入整数 n(1～3000):");                    //通过键盘输入确定整数 n
   scanf("%d",&n);
   c[1]=1;d[1]=2;
   c[2]=3;d[2]=5;
   kmax=1;                                              //数组与最大项序号赋初值
   for(i=3;i<=n;i++)
   {  for(k=c[i-1]+1;k<d[i-1];k++)
      {  t=0;                                           //k 枚举探求第 i 项分子 c
         for(j=1;j<i-1;j++)
            if(k==d[j]) {t=1;break;}
         if(t==0)
```

```
            {  c[i]=k;d[i]=k+i;                    //第 i 项分子 c、分母 d 赋值
               break;
            }
         }
         if(c[i]*d[kmax]>c[kmax]*d[i])
         kmax=i;                                   //比较得最大项的序号 kmax
      }
      printf("数列第%d项为: %ld/%ld。\n",n,c[n],d[n]);
      printf("数列前%d项中最大项为",n);
      for(i=1;i<=n;i++)                            //检查可能有多个最大项
        if(c[i]*d[kmax]==c[kmax]*d[i])
      printf("第%d项: %ld/%ld。\n",i,c[i],d[i]);
   }
```

4. 程序运行示例与分析

```
请输入整数 n(1～3000):2015
数列第 2015 项为: 3260/5275。
数列前 2015 项中最大项为第 1597 项: 2584/4181。
```

注意到每递推一项需用到前面的所有项,本案例递推的时间复杂度为 $O(n^2)$。

注意:本案例在求最大项时把分数比较转换为整数比较,以确保准确性,这一技巧是常用的。

顺便指出,上述分数的分子和分母构成的数对(1,2),(3,5),(4,7),(6,10),…常称为 Wythoff 对。Wythoff 对在数论和对策论中应用广泛。

3.4 幂 序 列

本节探讨双幂序列与幂积序列这两个涉及幂的典型案例的求解。

3.4.1 双幂序列

1. 案例提出

设 x、y 为非负整数,试计算集合

$$M=\{2^x,3^y\mid x\geqslant 0,y\geqslant 0\}$$

的元素由小到大排列的双幂序列第 n 项与前 n 项之和。

2. 递推设计

集合由 2 的幂与 3 的幂组成,实际上是给出两个递推关系。

设置一维数组 f,$f[k]$存储双幂序列的第 k 项。

显然,第 1 项也是最小项为 $f[1]=1$(当 $x=y=0$ 时)。

从第 2 项开始,为了实现从小到大排序,设置 a、b 两个变量,a 为 2 的幂,b 为 3 的幂,显然 $a\neq b$。

设置 k 循环($k=2,3,\cdots,n$,其中 n 为从键盘输入的整数),在 k 循环外赋初值:$a=2$,$b=3$。在 k 循环中通过比较选择赋值:

(1) 当 $a<b$ 时,由赋值 $f[k]=a$ 确定为序列的第 k 项;然后 $a=a*2$,即 a 按递推规律乘 2,为后一轮比较做准备。

(2) 当 $a>b$ 时，由赋值 $f[k]=b$ 确定为序列的第 k 项；然后 $b=b*3$，即 b 按递推规律乘 3，为后一轮比较做准备。

递推过程描述：

```
a=2;b=3;                          //为递推变量 a、b 赋初值
for(k=2;k<=n;k++)
{  if(a<b)
     { f[k]=a;a=a*2;}             //用 a 给 f[k]赋值
   else
     { f[k]=b;b=b*3;}             //用 b 给 f[k]赋值
}
```

在这一算法中，变量 a、b 是变化的，分别代表 2 的幂与 3 的幂。

为标注第 n 项为幂的形式，设置变量 t：$t=2$ 为 2 的幂，其指数为 p_2；$t=3$ 为 3 的幂，其指数为 p_3。

3. 程序实现

```
//双幂序列求解,c341
#include<stdio.h>
void main()
{  int k,n,t,p2,p3;
   double  a,b,s,f[100];
   printf("  求数列的第 n 项与前 n 项和,请输入 n: ");
   scanf("%d",&n);
   f[1]=1;p2=0;p3=0;
   a=2;b=3;s=1;
   for(k=2;k<=n;k++)
   {  if(a<b)
      {  f[k]=a;a=a*2;             //用 2 的幂给 f[k]赋值
         t=2;p2++;                 //t=2 表示 2 的幂,p2 为指数
      }
      else
      {  f[k]=b;b=b*3;             //用 3 的幂给 f[k]赋值
         t=3;p3++;                 //t=3 表示 3 的幂,p3 为指数
      }
      s+=f[k];
   }
   printf("  数列的第%d 项为: %.0f ",n,f[n]);
   if(t==2)                        //对输出项进行标注
       printf("(2^%d) \n",p2);
   else
       printf("(3^%d) \n",p3);
   printf("  数列的前%d 项之和为: %.0f \n",n,s);
}
```

4. 程序运行示例与分析

```
求数列的第 n 项与前 n 项和,请输入 n: 50
数列的第 50 项为: 1162261467 (3^19)
数列的前 50 项之和为: 3890875846
```

这一递推设计在一重循环中完成，时间复杂度与空间复杂度均为 $O(n)$。

变通：如果序列由 3 项幂或更多项幂组成，递推应如何实施？

3.4.2 幂积序列

1. 案例提出

设 x、y 为非负整数,试计算集合

$$M=\{2^x \cdot 3^y \mid x\geqslant 0, y\geqslant 0\}$$

的元素不大于指定整数 n 的个数,并求这些元素从小到大排序的第 m 项。

与双幂序列相比,幂积序列复杂在“积”字上,即幂积序列的项既可以是 2 的幂或 3 的幂,也可以是这两个幂之积。

下面在两个枚举设计求解基础上,给出案例的两个递推设计,并比较这 4 种设计的时间复杂度。

2. 递增枚举设计

1) 设计要点

集合元素由 2 的幂与 3 的幂及其乘积组成,设元素从小到大排序的第 k 项为 $f(k)$。显然,$f(1)=1$,$f(2)=2$,$f(3)=3$。

设置 a 循环,a 从 3 开始递增 1 至 n,对每一个 a(赋值给 j),逐次用 2 试商,然后逐次用 3 试商。试商后,若 $j>1$,说明原 a 有 2、3 以外的因数,不属于该序列;若 $j=1$,说明原 a 只有 2、3 的因数,属于该序列,把 a 赋值给序列第 k 项。

由于实施从小到大的试商和赋值,所得项无疑是从小到大的序列。

当 a 达到指定的整数 n 时,退出循环,输出指定项 $f(m)$。

2) 程序设计

```
//幂序列 2^x * 3^y 枚举求解,c342
#include<stdio.h>
void main()
{  int k,m;
   long  a,j,n,f[1000];
   printf("  计算不大于 n 的项数,请指定 n: ");
   scanf("%ld",&n);
   printf("  输出序列的第 m 项,请指定 m: ");
   scanf("%d",&m);
   f[1]=1;f[2]=2;k=2;
   for(a=3;a<=n;a++)
   {  j=a;
      while(j%2==0) j=j/2;            //反复用 2 试商
      while(j%3==0) j=j/3;            //反复用 3 试商
      if(j==1)
      { k++;f[k]=a;}                  //用 a 给 f[k]赋值
   }
   printf("  幂序列中不大于%ld 的项数为: %d\n",n,k);
   if(m<=k)
       printf("  从小到大排序的第%d 项为: %ld\n",m,f[m]);
   else
       printf("  所输序号 m 大于序列的项数!\n");
}
```

3）程序运行示例

```
计算不大于 n 的项数,请指定 n: 10000000
输出序列的第 m 项,请指定 m: 100
幂序列中不大于 10000000 的项数为: 190
从小到大排序的第 100 项为: 93312
```

3. 有针对性的枚举设计

上述枚举设计比较盲目,对大量含有 2、3 以外因数的整数也做了检测,有大量无效操作。为克服盲目性,下面给出一种有针对性的枚举设计,是一种带启发性的搜索设计,可大大提高枚举效率。

1）设计要点

为实现有针对性的枚举,设置 2 的幂与 3 的幂两个数组。

（1）t_2 数组,存储 2 的幂：$t_2[0]=2^0$，$t_2[1]=2^1$,…,$t_2[p_2-1]=2^{p_2-1}\leqslant n$,$t_2[p_2]>n$。

（2）t_3 数组,存储 3 的幂：$t_3[0]=3^0$，$t_3[1]=3^1$,…,$t_3[p_3-1]=3^{p_3-1}\leqslant n$,$t_3[p_3]>n$。

设置 i、j 二重枚举循环（i：$0\sim p_2-1$,j：$0\sim p_3-1$）,构造幂积语句为“t＝t2[i]＊t3[j];”。

（1）当 $i=0$ 时,t 为 3 的幂。

（2）当 $j=0$ 时,t 为 2 的幂。

（3）当 $i>0$ 且 $j>0$ 时,t 为 2 与 3 的幂积。

同时设置 f 数组,存储幂积 t。

（1）若 $t>n$,超出范围,不对 f 数组赋值。

（2）若 $t\leqslant n$,对 f 数组赋值语句为：“k＋＋;f[k]＝t;”。

通过以上构造幂有针对性地枚举,求出集合 M 中不大于指定整数 n 的所有 k 个元素。对这 k 个元素进行排序,以求得从小到大排序的第 m 项。

注意到集合 M 中不大于指定整数 n 的元素个数 k 的数量不会太大,采用较为简明的“逐项比较”排序法是可行的。实施排序中,从小到大排序到第 m 项即可,没有必要对所有 k 个元素排序。

2）程序设计

```
//有针对性的枚举,c343
#include<stdio.h>
void main()
{  int i,j,k,m,p2,p3;
   double d,n,t,t2[100],t3[100],f[10000];
   printf("  请指定 n,m: ");scanf("%lf,%d",&n,&m);
   t=1;p2=0;                                  //构造 2 的幂数组
   while(t<=n) {t=t*2;p2++;t2[p2]=t;}
   t=1;p3=0;                                  //构造 3 的幂数组
   while(t<=n) {t=t*3;p3++;t3[p3]=t;}
   t2[0]=t3[0]=1;k=0;
   for(i=0;i<=p2-1;i++)
   for(j=0;j<=p3-1;j++)
   {  t=t2[i]*t3[j];                          //构造幂积 t 并进行检测和赋值
```

```
        if(t<=n) { k++;f[k]=t;}
    }
    printf("  幂积序列中不大于%.0f 的项数为: %d\n",n,k);
    if(m<=k)
    {  for(i=1;i<=m;i++)                    //逐项比较,排序至第 m 项
       for(j=i+1;j<=k;j++)
          if(f[i]>f[j]) { d=f[i];f[i]=f[j];f[j]=d;}
       printf("  从小到大排序的第%d 项为: %.0f\n",m,f[m]);
    }
    else
       printf("  所输入的 m 大于序列的项数!\n");
}
```

3) 程序运行示例

```
请指定 n,m: 1000000000,300
幂序列中小于 1000000000 的项数为: 306
从小到大排序的第 300 项为: 774840978
```

4. 递推排序设计

为了进一步提高搜索效率,试应用递推进行求解。

1) 设计要点

(1) 确定递推关系。

为探索 $x+y=i$ 时各项与 $x+y=i-1$ 时各项之间的递推规律,剖析 $x+y$ 的前几个值情形:

$x+y=0$ 时,元素为 1(初始条件)。

$x+y=1$ 时,元素为 $2\times1=2$,$3\times1=3$,共 2 项。

$x+y=2$ 时,元素有 $2\times2=4$,$2\times3=6$,$3\times3=9$,共 3 项。

$x+y=3$ 时,元素有 $2\times4=8$,$2\times6=12$,$2\times9=18$,$3\times3\times3=27$,共 4 项。

……

可归纳出以下递推关系:$x+y=i$ 时,序列共 $i+1$ 项,其中前 i 项是 $x+y=i-1$ 时的所有 i 项分别乘 2 所得,最后一项为 $x+y=i-1$ 时的最后一项乘 3 所得(即 $t=3^i$)。

注意,对 $x+y=i-1$ 的所有 i 项分别乘 2,设为 $f[h]*2$,必须检测其是否小于 n 而大于 0。同样,对 t 也必须检测是否小于 n 而大于 0。只有小于 n 且大于 0 时才能赋值。

这里要指出,最后若干行可能不是完整的,即可能只有前若干项能递推出新项。为此设置变量 u,当一行有递推项时 $u=1$,否则 $u=0$。当 $u=0$ 时停止,否则会影响序列的项数。

(2) 以 $n=1000$ 为例,具体说明递推的实施。

```
      f(1)=1
i=1: f(2)=2    f(3)=3
i=2: f(4)=4    f(5)=6    f(6)=9
i=3: f(7)=8    f(8)=12   f(9)=18   f(10)=27
i=4: f(11)=16  f(12)=24  f(13)=36  f(14)=54  f(15)=81
i=5: f(16)=32  f(17)=48  f(18)=72  f(19)=108 f(20)=162 f(21)=243
i=6: f(22)=64  f(23)=96  f(24)=144 f(25)=216 f(26)=324 f(27)=486 f(28)=729
i=7: f(29)=128 f(30)=192 f(31)=288 f(32)=432 f(33)=648 f(34)=972
i=8: f(35)=256 f(36)=384 f(37)=576 f(38)=864
i=9: f(39)=512 f(40)=768
```

每一列的下一个数是上一个数的 2 倍，而每一行的最后一个数为 3 的幂。

（3）对所产生的项排序。

当所有递推项完成后，对所有 k 项应用逐项比较进行从小到大排序。排序后输出指定的第 m 项。

2）程序实现

```
//递推排序求解,c344
#include<stdio.h>
void main()
{  int i,j,h,k,m,u,c[100];
   double d,n,t,f[1000];
   printf("  计算小于 n 的项数,请指定 n: "); scanf("%lf",&n);
   printf("  输出序列的第 m 项,请指定 m: "); scanf("%d",&m);
   k=1;t=1.0; i=1;
   c[0]=1; f[1]=1.0;
   while(1)
   {  u=0;
      for(j=0;j<=i-1;j++)
      {  h=c[i-1]+j;
         if(f[h]*2<n && f[h]>0)                    //第 i 行各项为前一行各项乘 2
         {  k++;f[k]=f[h]*2;u=1;
            if(j==0) c[i]=k;                       //该行的第 1 项的项数值赋给 c(i)
         }
         else break;
      }
      t=t*3;                                       //最后一项为 3 的幂
      if(t<n && t>0)
        { k++;f[k]=t; }                            //用 t 给 f[k]赋值
        if(u==0) break;
        i++;
   }
   for(i=1;i<k;i++)                                //逐项比较排序
     for(j=i+1;j<=k;j++)
       if(f[i]>f[j])
         { d=f[i];f[i]=f[j];f[j]=d;}
   printf("  幂序列中小于%f 的项数为: %d\n",n,k);
   if(m<=k)
       printf("  从小到大排序的第%d 项为: %.0f\n",m,f[m]);
   else
       printf("  所输入的 m 大于序列的项数!\n");
}
```

3）程序运行示例

```
计算小于 n 的项数,请指定 n: 1000000000000
输出序列的第 m 项,请指定 m:500
幂序列中不大于 1000000000000 的项数为: 534
从小到大排序的第 500 项为: 391378894848
```

5. 递推结合比较赋值设计

上述递推在排序上占用大量时间，递推结合比较赋值可省去排序操作，进一步降低复杂度。

1) 设计要点

从 $u=1$，$f(u)=1$ 开始,在已求得 $f(u)$的基础上,可递推求出 $f(u+1)$：求出各大于 $f(u)$的最小数,取其中最小者即为 $f(u+1)$。

递推结合比较赋值设置永真外循环,实施乘 2 的内循环。

首先,从 $p=0$ 开始,若 $q[p]\leqslant f[u]$,则递推得一个 3 的幂,即 $q[p]=3^p$,并赋给最小值标志量 h。

其次,转入内循环 $i(0\sim p-1)$中,若 $q[i]\leqslant f[u]$,则 $q[i]$乘 2。

再次,$q[i]$与 h 比较,即 $2^j\times 3^i(i<p)$与 3^p 比较,取较小者为 h。

(1) 若 $h\leqslant n$,则 h 赋值给序列新的项,用 u 标记项数。

(2) 若 $h>n$,表明递推结合比较赋值完成,退出外循环,输出序列的项数 u 与序列中指定的项 $f[m]$后结束。

2) 程序设计

```
//求 3^i * 2^j<n 的项数及从小到大第 m 项,c345
#include<stdio.h>
void main()
{  int i,m,u,p;
   double n,h,f[1000],q[100];
   printf("  计算小于 n 的项数,请指定 n: ");
   scanf("%lf",&n);
   printf("  输出序列的第 m 项,请指定 m: ");
   scanf("%d",&m);
   u=1; f[u]=1.0;
   p=0; q[p]=1.0;
   while(1)
   {  if(q[p]<=f[u])
      { p++; q[p]=3*q[p-1]; }
      h=q[p];                          //递推 3 的幂,q[p]=3^p
      for(i=0;i<p;i++)
      {  if(q[i]<=f[u]) q[i]*=2;  //幂积 q[i]=2^j*3^i,j=1,2,3,…
         if(q[i]<h) h=q[i];
      }
      if(h>n) break;
      u++;f[u]=h;
   }
   printf("  幂序列中小于%.0f 的项数为: %d\n",n,u);
   if(m<=u)
       printf("  从小到大排序的第%d 项为: %.0f\n",m,f[m]);
   else
       printf("  所输入的 m 大于序列的项数!\n");
}
```

3) 程序运行示例

```
计算小于 n 的项数,请指定 n: 10000000000000
输出序列的第 m 项,请指定 m: 600
幂序列中小于 10000000000000 的项数为: 624
从小到大排序的第 600 项为: 5355700839936
```

思考：如果需对显示的第 m 项标注表达式,即需标注表达式中的 2 与 3 的幂指数,程序

应如何修改？

6. 4 个算法的时间复杂度比较

当 n 很大时，递推求解的速度明显快于枚举。

1）递增枚举复杂度分析

枚举算法简单明了，对整数 a 操作，每一个 a 进行连续除 2、除 3 操作，平均估算为 10 次，整个 n 个数的操作为 $10n$ 次，算法复杂度为 $O(n)$。

2）有针对性枚举复杂度分析

有针对性枚举的双重循环次数要小于排序的双重循环次数。当 n 充分大时，如果按项数 $k<\sqrt[4]{n}$ 估计，该算法的排序频数小于 mk，因而算法的时间复杂度低于 $O(\sqrt{n})$。

3）递推排序算法复杂度分析

递推排序算法对序列项数进行操作，当 n 充分大时，项数 $k<\sqrt[4]{n}$。其中逐项比较排序的时间复杂度是 $O(n^2)$，即 $O(k^2)$，因而递推排序算法的时间复杂度为 $O(\sqrt{n})$。

4）递推结合比较赋值算法复杂度分析

外循环次数为序列项数 u，当 n 充分大时，项数 $u<\sqrt[4]{n}$。内循环 $i(0\sim p-1)$ 中 p 为幂指数，$3^p=n$，即 $p=\log_3 n$。注意到 p 是从 0 增长的，每一项按平均即 $p/2$ 估算，因而得递推结合比较赋值算法复杂度为 $O(\sqrt[4]{n}\log_3\sqrt{n})$。

当 n 充分大时，有 $\sqrt[4]{n}\log_3\sqrt{n}<\sqrt{n}<n$。可见以上 4 种算法中，递推结合比较赋值算法时间最短，而递增枚举所需时间最长。

由两个递推求解可见，递推设计的难点在于如何准确地寻找并确定递推关系。

变通：如何改进设计以进一步实现求解三幂积序列？

3.5　数阵与网格

本节应用递推探讨两个典型的数阵案例，一个是著名的杨辉三角，另一个是新颖的交通方格网。

3.5.1　杨辉三角

1. 案例背景

杨辉三角历史悠久，是我国古代数学家杨辉揭示二项展开式各项系数的数字三角形。

我国北宋数学家贾宪约 1050 年首先使用“贾宪三角”进行高次开方运算，南宋数学家杨辉在《详解九章算法》中记载并保存了“贾宪三角”，故后世称杨辉三角。元朝数学家朱世杰在《四元玉鉴》中扩充了“贾宪三角”。在欧洲直到 1623 年以后，法国数学家帕斯卡才发现了与杨辉三角类似的“帕斯卡三角”。

杨辉三角构建规律主要包括横行各数之间的大小关系以及不同横行数字之间的联系，奥妙无穷：每一行的首尾两数均为 1；第 k 行共 k 个数，除首尾两数外，其余各数均为上一行的肩上两数的和。图 3-1 为 5 行杨辉三角。

设计程序，构造并输出杨辉三角的前 n 行（n 从键盘输入）。

2. 应用数组递推设计

1) 设计要点

考察杨辉三角的构建规律,第 i 行有 i 个数,其中第 1 个数与第 i 个数都是 1,其余各项为它的肩上两数之和(即上一行中相应项及其前一项之和)。

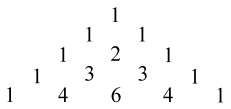

图 3-1 5 行杨辉三角

设置二维数组 $a(n,n)$,根据构成规律实施递推,递推关系如下:

$$a(i,j)=a(i-1,j-1)+a(i-1,j) \quad (i=3,4,\cdots,n;j=2,3,\cdots,i-1)$$

初始值:$a(i,1)=a(i,i)=1$ $(i=1,2,\cdots,n)$。

为了输出左右对称的等腰数字三角形,设置二重循环:设置 i 循环控制打印 n 行;每一行开始先打印 $40-3i$ 个前导空格,然后设置 j 循环控制打印第 i 行的各数组元素 $a(i,j)$。

2) 程序设计

```
//杨辉三角,c351
#include<stdio.h>
void main()
{  int n,i,j,k,a[20][20];
   printf("  请输入行数 n: ");
   scanf("%d",&n);
   for(i=1;i<=n;i++)
     {a[i][1]=1;a[i][i]=1;}              //确定初始条件
   for(i=3;i<=n;i++)
   for(j=2;j<=i-1;j++)                   //递推实施
       a[i][j]=a[i-1][j-1]+a[i-1][j];
     for(i=1;i<=n;i++)                   //控制输出 n 行
     {  for(k=1;k<=40-3*i;k++)
          printf(" ");                   //控制输出第 i 行的前导空格
        for(j=1;j<=i;j++)
          printf("%6d",a[i][j]);         //控制输出第 i 行的 i 个元素
        printf("\n");
     }
}
```

3) 程序运行示例

运行程序,输入 $n=10$,则打印 10 行杨辉三角,如图 3-2 所示。

```
                    1
                  1   1
                1   2   1
              1   3   3   1
            1   4   6   4   1
          1   5  10  10   5   1
        1   6  15  20  15   6   1
      1   7  21  35  35  21   7   1
    1   8  28  56  70  56  28   8   1
  1   9  36  84 126 126  84  36   9   1
```

图 3-2 10 行的杨辉三角

3. 应用变量迭代设计

1) 设计要点

杨辉三角实际上是二项展开式各项的系数,即第 $n+1$ 行的 $n+1$ 个数分别是从 n 个元素中取 $0,1,2,\cdots,n$ 个元素的组合数 $C(n,0),C(n,1),\cdots,C(n,n)$。注意到组合公式

$$C(n,0)=1$$

$$C(n,k)=(n-k+1)/k\times C(n,k-1)\quad (k=1,2,\cdots,n)$$

根据这一递推规律，可不用数组，直接应用变量迭代求解。

2）程序设计

```
//应用变量迭代求解,c352
#include<stdio.h>
void main()
{  int m,n,cnm,k;
   printf("  请输入行数 n: ");
   scanf("%d",&n);
   for(k=1;k<=40;k++) printf(" ");
   printf("%6d\n",1);                    //输出第 1 行的 1
   for(m=1;m<=n-1;m++)
   {  for(k=1;k<=40-3*m;k++)
      printf(" ");
      cnm=1;
      printf("%6d",cnm);                 //输出每行开始的 1
      for(k=1;k<=m;k++)
      {  cnm=cnm*(m-k+1)/k;              //计算第 m 行的第 k 个数
         printf("%6d",cnm);
      }
      printf("\n");
   }
}
```

由以上两个不同的递推可以看到，递推方式并不是一成不变的，往往有多种方式可供选择。

本案例的递推设计与迭代设计的时间复杂度均为 $O(n^2)$。

3.5.2　交通方格网

1. 案例提出

某市城区的交通方格网如图 3-3 所示，城区中一座山占据了交通网中(3,2)、(4,2)与(4,3)，即这 3 个交叉点尚未开通，另有从(2,3)至(2,4)、(5,0)至(6,0)的两条打“×”标记的路段正在维护，禁止通行。

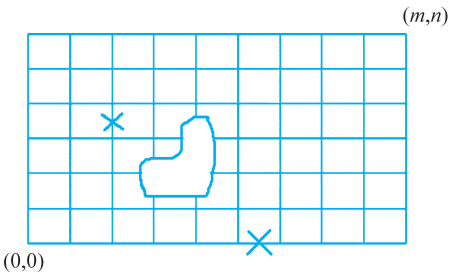

图 3-3　交通方格网示意图

试统计从始点(0,0)到终点(m,n)的不同最短路线(路线中各段只能从左至右、从下至上，不走回头路)的条数。

输入正整数 m、n($m\leqslant 20$,$n\leqslant 20$),输出从始点(0,0)到终点(m,n)的最短路线的条数。

2. 设计要点

没有障碍的交通方格网,每一条路线共 $m+n$ 段,其中横向 m 段,纵向 n 段,每一条不同路线对应从 $m+n$ 个元素中取 m 个元素(以放置横向段)的组合数。

因而不同路线条数为

$$C_{m+n}^{m}=\frac{n+1}{1}\times\frac{n+2}{2}\times\cdots\times\frac{n+m}{m}$$

今交通方格网中设置了诸多障碍,试应用递推设计求解。

设 $f(x,y)(0<x\leqslant m,0<y\leqslant n)$ 为从始点(0,0)到点(x,y)的不同最短路线的条数。注意到最短路线中点(x,y)的前一个点为($x-1$,y)与(x,$y-1$)这两个,则有以下几点。

1) 递推关系

$$f(x,y)=f(x-1,y)+f(x,y-1)$$

2) 边界条件

$$f(x,0)=1\ (0<x\leqslant 5)$$
$$f(x,0)=0\ (5<x\leqslant m)$$
$$f(0,y)=1\ (0<y\leqslant n)$$

3) 障碍处理

(1) 城区的一座山占据网中的(3,2)、(4,2)、(4,3)3 个交叉点,可令

$$f(3,2)=f(4,2)=f(4,3)=0$$

(2) 从(2,3)至(2,4)段禁止通行,则对 $f(2,4)$的赋值只有 $f(1,4)$,即

$$f(2,4)=f(1,4)$$

3. 程序实现

```
//带障碍的交通路线问题,c353
#include<stdio.h>
void main()
{  int m,n,x,y; long f[21][21];
   printf("  请输入正整数 m,n: "); scanf("%d,%d",&m,&n);
   for(x=1;x<=5;x++) f[x][0]=1;              //确定递推边界条件
   for(x=5;x<=m;x++) f[x][0]=0;              //边界维护路段处理
   for(y=1;y<=n;y++) f[0][y]=1;
   for(x=1;x<=m;x++)
     for(y=1;y<=n;y++)                       //实施递推
       if(x==3 && y==2 || x==4 && y==2 || x==4 && y==3)
         f[x][y]=0;                          //山所占据的 3 点处理
       else if(x==2 && y==4)
         f[x][y]=f[x-1][y];                  //中间维护路段处理
       else
         f[x][y]=f[x-1][y]+f[x][y-1];        //其他点递推
   printf("  最短路线条数为: %ld \n",f[m][n]);
}
```

4. 程序运行示例与分析

```
请输入正整数 m,n: 15,15
最短路线条数为: 48882820
```

本问题的难点在于对网格中的障碍分类处理，算法在递推中就“山所占据的 3 点”“中间维护路段”与“边界维护路段”3 类障碍分别实施不同的赋值处理。

递推算法在二重循环中实现，时间复杂度为 $O(mn)$。

3.6　整数划分问题

正整数 s（简称为和数）的划分（又称分划）是把 s 分成为若干正整数（简称为零数或部分）之和，划分式中允许零数重复，且不考虑零数的次序。

求 $s=12$ 共有多少个不同的划分式？展示出所有这些划分式。

3.6.1　整数划分递推设计

1. 探索划分的递推关系

难点在于展示出所有这些划分式，为避免重复，约定划分式中零数按不降排列。

为了建立递推关系，先对和数 k 较小时的划分式进行观察归纳：

$k=2$：1+1;2

$k=3$：1+1+1;1+2;3

$k=4$：1+1+1+1;1+1+2;1+3;2+2;4

$k=5$：1+1+1+1+1;1+1+1+2;1+1+3;1+2+2;1+4;2+3;5

由以上各划分看到，除和数本身 $k=k$ 这一特殊划分式外，其他每一个划分式至少为两项之和。和数 k 的划分式与和数 $k-1$ 的划分式存在以下递推关系。

（1）在所有和数 $k-1$ 的划分式前加一个零数 1 都是和数 k 的划分式。

（2）和数 $k-1$ 的划分式的前两个零数做比较，如果第 1 个零数 x_1 小于第 2 个零数 x_2，则把第 1 个零数加 1 后成为和数 k 的划分式。

2. 递推设计

设置三维数组 a，$a(k,j,i)$为和数 k 的第 j 个划分式的第 i 个数。

从 $k=2$ 开始，显然递推的初始条件为

$$a(2,1,1)=1,\quad a(2,1,2)=1,\quad a(2,2,1)=2$$

根据递推关系，实施递推。

（1）实施在 $k-1$ 所有划分式前加 1 操作：

```
a(k,j,1)=1;
for(t=2;t<=k;t++)
    a(k,j,t)=a(k-1,j,t-1);          //k-1 的第 t-1 项变为 k 的第 t 项
```

（2）若 $k-1$ 划分式第 1 项小于第 2 项，第 1 项加 1，变为 k 的第 i 个划分式：

```
if(a(k-1,j,1)<a(k-1,j,2))
{  a(k,i,1)=a(k-1,j,1)+1;
   for(t=2;t<=k-1;t++)
     a(k,i,t)=a(k-1,j,t);
}
```